Natural Resources for Human Health
eISSN: 2583-1194

Original Research

Received 29 May 2026
Revised 21 June 2026
Accepted 18 July 2026

Natural Resources for Human Health 2026, Vol. 6 (10s): 929–943

KEYWORDS:
open-source geospatial libraries; GIS; spatial data analysis; classification framework; PRISMA; Python; R; C/C++; Rust; Julia

Cross-Language Classification of Open-Source Geospatial Libraries for Natural Resource Management: A Systematic Review and Framework

Ranga Rao Velamala*[1], Pallavi Bhatt[2,3], N.S.R. Prasad[4], A. Simhachalam[5], N.S. Gahlod[6], Sanjay A. Dhale[7]

[1]Professor of Practice, Department of Computer Science and Engineering, Visakha Institute of Engineering & Technology (A), Visakhapatnam, affiliated with Jawaharlal Nehru Technological University–Gurajada, Vizianagaram, Andhra Pradesh, India.
[2]Assistant Professor, Graphic Era Hill University, Dehradun, Uttarakhand, India
[3]Assistant Professor, Graphic Era (Deemed) University, Dehradun, Uttarakhand, India
[4,5]Assistant Professor, Centre for Geoinformatics Application in Rural Development, National Institute of Rural Development and Panchayati Raj, Ministry of Rural Development, Government of India, Rajendranagar, Hyderabad, Telangana, India
[6]Soil Survey Officer, Soil and Land Use Survey of India, Department of Agriculture and Farmers Welfare, Ministry of Agriculture and Farmers Welfare, Government of India, Nagpur, Maharashtra, India
[7]Soil Survey Officer, Soil and Land Use Survey of India, Department of Agriculture and Farmers Welfare, Ministry of Agriculture and Farmers Welfare, Government of India, Hyderabad, Telangana, India
*Corresponding Author, Email: velamala.ranga@gmail.com

ABSTRACT: Geographical Information Systems (GIS) have traditionally been implemented through desktop environments such as QGIS. The increasing availability of open-source geospatial libraries across multiple programming languages enables spatial data analysis through programmatic workflows. This study systematically examines twelve programming-language environments (C/C++, Python, R, Julia, Java, Scala, JavaScript, TypeScript, Go, Rust, Kotlin and Swift), together with SQL/Spatial SQL (treated here as a query and data-management paradigm), and their associated open-source geospatial libraries. Functionality, interoperability, and ecosystem maturity are evaluated through a structured synthesis of the literature. After systematically screening 1,245 records, 370 sources met the inclusion criteria and were included in the analysis. Of these, 210 focused on Python/R, 85 on C/C++/Java, 45 on JavaScript/TypeScript, and 30 on emerging languages (Rust, Julia, Go). The Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework was proposed as a structured, language-oriented classification that organises libraries according to programming language and primary GIS function. Identified functional categories include analytical processing, core computational support, distributed geospatial processing, statistical modelling, web-based visualisation, performance optimisation, scientific computing, infrastructure services and mobile GIS applications. The reviewed literature indicates strong representation of Python in integrative analytical workflows, the foundational role of C and C++ libraries in computational geometry and data processing, the importance of JVM-based frameworks for distributed geospatial analytics, and the emerging applicability of Rust and Julia in high-performance spatial computing contexts. The LOGLE framework organises libraries

Natural Resources for Human Health
eISSN: 2583-1194

by programming language, functional role, and system layer. Building on existing single-language surveys and OSGeo ecosystem reviews, it consolidates fragmented documentation into a coherent classification. The framework is conceptually scoped and is intended to support future empirical validation and benchmarking studies.

1. INTRODUCTION

Geographic Information Systems (GIS) provide frameworks for managing and analysing spatial data. Geospatial libraries deliver computational tools for executing these operations programmatically. Open-source geospatial libraries now span multiple programming languages and extend spatial data science capabilities in remote sensing, statistical modelling, interactive visualisation, and large-scale computation. Their adoption in urban analytics and environmental monitoring increases methodological transparency and research accessibility (Coetzee et al., 2020; Mobasheri et al., 2020). These libraries are also widely applied in natural resource management applications, including soil and land resource mapping (Kanwar et al., 2022; Kumar et al., 2018), land degradation and landscape-metric assessment (Zaragozí et al., 2012), terrain and geomorphometric analysis (Ilich et al., 2023), and water resources management (Chen et al., 2010). Their zero licensing cost makes them particularly valuable to research and management institutions in developing and under-resourced countries, where proprietary GIS software represents a significant financial barrier to natural resource assessment (Chen et al., 2010; Kumar et al., 2018).

The open-source geospatial software ecosystem is fragmented, with libraries developed for specific programming languages and functional domains. Python, R, C/C++, Java, Scala, JavaScript, Rust, Julia, TypeScript, Go, Kotlin, and Swift support geospatial libraries with distinct performance characteristics. Existing literature addresses individual tools or single-language environments but does not provide a comprehensive cross-language perspective (Duarte & Teodoro, 2021; Rosenbaum et al., 2020). Low-level libraries in C and C++ form the core infrastructure: GDAL/OGR, GEOS, and PROJ perform raster and vector processing, coordinate reference system management, and computational geometry (Warmerdam, 2008a), underpinning higher-level libraries and maintaining interoperability and computational consistency (Pebesma, 2018; Rosenbaum et al., 2020). High-level ecosystems are particularly well developed in Python and R. Python functions as a primary environment for geospatial analytics through its scientific computing ecosystem, supporting integrative workflows for reproducible spatial analysis and spatial econometrics (Boeing, 2017a, 2017b, 2025; Jordahl et al., 2023; Rey & Anselin, 2010; Rey et al., 2022). R remains a standard environment for geostatistics and ecological modelling through sf, terra, gstat, spdep, tmap, and lidR (Bivand et al., 2013; Bivand & Wong, 2018; Hijmans, 2023; Pebesma, 2018; Tennekes, 2018). Java provides GeoTools and the JTS Topology Suite for geospatial processing and computational geometry (Johansson & Harrie, 2005; Shekhar et al., 2017; Srivastava, 2012; Turton, 2008). JavaScript provides Leaflet and Turf.js for web-based visualisation (Agafonkin, 2019; Netek et al., 2023). JVM frameworks, including Apache Sedona, GeoMesa, and GeoTrellis, enable scalable spatio-temporal processing (Haynes et al., 2020; Yu et al., 2019). Julia and Rust support high-performance computation, and Go supports cloud-native systems. Despite extensive documentation of individual libraries and language-specific ecosystems, integrated examination across languages, functional roles, and maturity characteristics remains limited.

The open-source geospatial software ecosystem has expanded over the last two decades (Coetzee et al., 2020; Mobasheri et al., 2020; Rosenbaum et al., 2020). Proliferation across languages and domains has increased fragmentation, complicating tool selection, integration, and application within reproducible workflows. Capabilities are unevenly distributed across language environments, constraining consistency in tool use and workflow design. Scholarly attention to the influence of programming-language choice on analytical performance, interoperability, and workflow efficiency remains insufficient (Rosenbaum et al., 2020). The lack of a structured basis for comparison may contribute to ad-hoc tool selection, inefficient toolchains, reduced reproducibility, and variable analytical outcomes.

A critical review of the existing literature identifies three specific gaps. First, existing surveys catalogue tools within single-language ecosystems (Coetzee et al., 2020; Mobasheri et al., 2020), limiting assessment of functional equivalence, analytical capability, and interoperability across languages. Second, no unified classification organises libraries by programming language and functional role within analytical workflows, leaving overlaps and discontinuities across data acquisition, processing, analysis, and visualisation unexamined. Third, empirical evidence on cross-language adoption remains limited and uneven, with comparative work concentrating on Python and R and providing less coverage of Julia, JavaScript, and C++. These gaps indicate the need for a structured, evidence-based framework for cross-language comparison and tool selection.

The present study addresses these gaps by introducing the Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework, a structured taxonomy and conceptual synthesis for classifying open-source geospatial libraries across programming languages. Building on the community-driven development and open standards fostered by the Open Source Geospatial Foundation (OSGeo) (Coetzee et al., 2020), the framework is applied to examine how libraries across programming languages support spatial data analysis in programmatic workflows, classifying libraries by language and functional role, and delineating their capabilities and limitations across application domains such as urban studies, environmental assessment, participatory mapping, and remote sensing (Mobasheri et al., 2020). The selection of environments and libraries follows explicit criteria. Environments are included when geospatial libraries are documented in the literature and demonstrate established use in geospatial workflows. Libraries are included when they support core functions in data processing, analysis, or visualisation and when evidence of their application is documented; libraries lacking such evidence are excluded.

Accordingly, the study pursues five objectives: (i) to synthesise peer-reviewed literature, authoritative project documentation, and adoption indicators to characterise usage patterns of open-source geospatial libraries across programming languages; (ii) to propose the LOGLE framework as a structured, multi-layer taxonomy that classifies libraries by programming language, functional role, and system layer; (iii) to identify cross-language disparities in library availability, functional coverage, and documented research adoption; (iv) to derive recommendations for tool selection and workflow integration to support reproducibility, interoperability, and methodological consistency; and (v) to establish a structured classification that can underpin future empirical benchmarking of cross-language geospatial performance and reproducibility.

2. MATERIALS AND METHODS

This study employs a systematic review methodology to collect, analyse, and synthesise literature and documented usage metrics related to open-source geospatial libraries across multiple programming languages. The methodology follows established systematic review protocols (Grant & Booth, 2009; Kitchenham & Charters, 2007) and adheres to PRISMA 2020 reporting standards (Page et al., 2021). It comprises four main phases: search strategy, inclusion and exclusion criteria, evidence extraction and coding, and synthesis and framework construction. A flow diagram of the study selection process is presented in Figure 1.

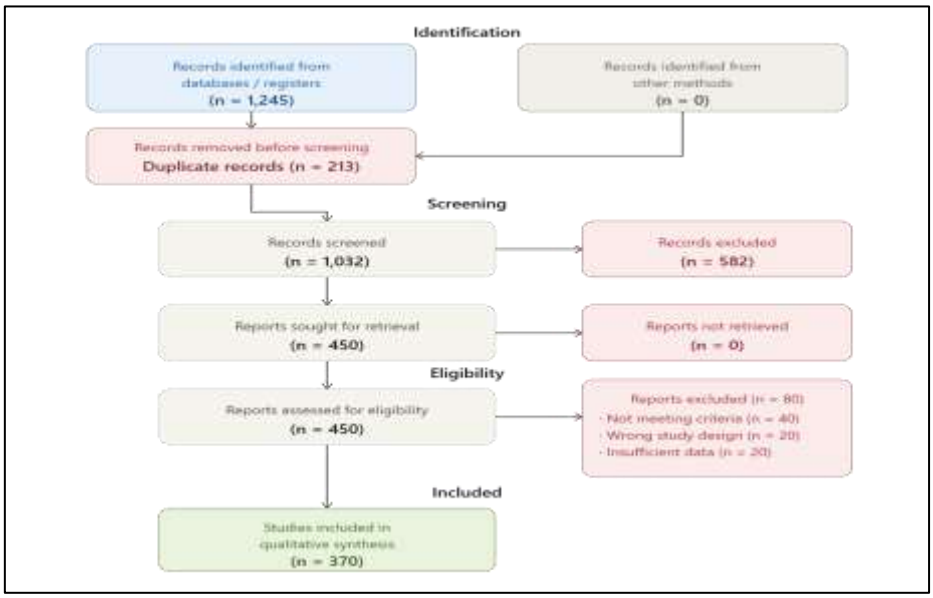

Figure 1. Flow diagram of the study selection process for the systematic review of open-source geospatial libraries across programming languages

2.1 Search Strategy

Comprehensive searches were conducted across Scopus, Web of Science, IEEE Xplore, Google Scholar, GitHub, GitLab, and official project websites up to March 2026. The representative query structure was: ("Python" OR "R" OR "JavaScript" OR "C++" OR "Java" OR "Rust" OR "Julia" OR "Go") AND ("geospatial library" OR "GIS library" OR "spatial analysis" OR

"geospatial processing" OR "spatial data"). Searches were conducted iteratively, with data extraction and synthesis performed in parallel under a predefined protocol. All cited works dated 2025–2026 were verified as publicly available at the time of writing.

2.2 Inclusion and Exclusion Criteria

Eligibility criteria were established to ensure alignment with the adoption, functionality, and comparative evaluation of open-source geospatial libraries. Inclusion criteria comprised peer-reviewed articles reporting the use, evaluation, or integration of geospatial libraries; technical reports and official documentation describing core framework architectures; empirical studies reporting performance metrics, adoption patterns, or comparative evaluations; and publicly available code repositories providing verifiable implementation evidence. Studies focused exclusively on proprietary software were excluded, as were publications lacking empirical evidence or technical substantiation. Programming languages were retained for comparative analysis only when their geospatial ecosystems demonstrated sufficient maturity. Maturity was operationally defined as the presence of at least three open-source libraries that (a) support core geospatial functions, (b) exhibit repository activity (commits or issue responses) within the 24 months preceding the search, and (c) are documented in at least one peer-reviewed or technical publication. Languages failing to meet these criteria were excluded. The criteria were applied uniformly across all candidate languages at the time of data collection.

Database and register searches yielded 1,245 records. After the removal of 213 duplicates, 1,032 records remained for title and abstract screening. Of these, 450 records progressed to full-text assessment. Full-text screening resulted in the exclusion of 80 studies due to methodological irrelevance (n = 40), unsuitable study design (n = 20), and insufficient data availability (n = 20), yielding a final set of 370 studies (Figure 1). To enhance methodological transparency, a random 20 % subset of the 1,032 records remaining after deduplication (n = 206) was independently re-screened by a second reviewer following a four-week interval. Inter-rater agreement was high (Cohen's κ = 0.87). Remaining discrepancies were resolved through joint re-examination of the original sources and consensus discussion. Full search strings for each database, the complete list of the 370 included sources, and the coding spreadsheet employed for functional classification and maturity rating are provided as Supplementary Material.

2.3 Evidence Extraction and Coding

Data were systematically extracted from the included sources and organised in a structured database. The extracted information covered the programming language or runtime ecosystem, the name and version of each library, its primary functional role within geospatial workflows, and the application context in which it was used. Indicators of adoption, such as repository activity and citation patterns, were examined where publicly available.

2.4 Synthesis and LOGLE Framework Construction

Evidence synthesis was conducted using narrative thematic analysis following Braun and Clarke (2006) and established qualitative traditions in geographic research (Cope, 2010), adapted for systematic review contexts to enable structured cross-language comparison. This involved iterative identification of themes across programming language ecosystems and functional library categories, progressing from initial coding to the definition of classificatory themes aligned with geospatial workflow stages. The Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework was developed by systematically mapping programming language ecosystems against extracted library attributes, functional roles, and documented use cases. Classification rules were defined based on alignment between library functionality and the four workflow stages of data acquisition, processing, analysis, and visualisation. Libraries with multi-functional capabilities were assigned to all relevant stages based on documented evidence of use. Maturity ratings reported in Table 1 are qualitative ordinal assessments. An explicit scoring rubric based on four equally weighted criteria (years since first stable release, sustained repository activity in the preceding 24 months, scholarly citation counts, and documented production or enterprise use) was applied consistently.

3. THE LOGLE FRAMEWORK

Table 1 & 2 presents the Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework, which classifies open-source geospatial computing technologies across three explicitly defined layers: (i) programming languages and runtime ecosystems, (ii) geospatial software libraries and frameworks, and (iii) foundational geospatial data systems. This layered structure enables consistent comparison of technologies operating at different levels of abstraction within the geospatial computing stack. The framework incorporates ecosystem maturity, represented by year of introduction and a qualitative

assessment of long-term production stability, together with primary GIS application domains (Desktop GIS, Web GIS, Mobile GIS), to support informed tool selection across deployment environments.

Table 1. The Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework: classification of open-source geospatial technologies by language/ecosystem, year of introduction, stable baseline, primary functional role, and maturity (Very High–Emerging). Rows 11 and 12 represent cross-cutting categories (Cloud-Native Platforms and OGC Standards).

#	Technology Family	Language / Ecosystem	Introduced	Stable Baseline	Primary Functions	Maturity *	Scholarly / Technical Foundations
1	C/C++ Core Engines	C / C++	1972/1985	C23/C++20+	Geometry computation, spatial engines, coordinate systems	Very High	Warmerdam (2008a, 2008b); Neteler & Mitasova (2008)
2	Spatial SQL & Indexing	SQL/ Spatial SQL	1974	SQL:2023	Spatial databases, querying, indexing	Very High	Obe & Hsu (2021); ISO (2006); Güting (1994)
3	Python Data Science	Python	1991	3.x	Geospatial data science, GeoAI/spatial ML, raster/vector processing	High	Jordahl et al. (2020); McKinney (2010); Hoyer & Hamman (2017); Zhu et al. (2017); Rey et al. (2020)
4	R Spatial Statistics	R	1993	4.x	Spatial statistics, geostatistics, ecological modelling	High	Pebesma (2018); Bivand et al. (2013); Hijmans et al. (2020); Roussel et al. (2020); Pebesma & Bivand (2023)
5	Julia Simulation	Julia	2012	1.x	Numerical geospatial simulation, earth-system modelling	Emerging	Hoffimann et al. (2018); Perkel (2019); JuliaEarth (2023)
6	Java/Scala Enterprise	Java/Scala	1995/2004	Java 17/21	Enterprise GIS, SDI, distributed spatial systems	High	Ježek (2006); Huang et al. (2017); Aji et al. (2013);

							Hughes et al. (2015)
7	Go Services	Go	2009	1.x	Cloud-oriented GIS services and APIs	Medium	Zhang et al. (2019)
8	Rust Systems	Rust	2010	1.x	High-performance spatial computing, WASM GIS	Emerging	GeoRust Project (2026)
9	JavaScript/TypeScript Web	JavaScript/ TypeScript	1995/2012	ES2023+	Web mapping, visualisation, client-side GIS	Very High	Goodchild (2007); Haklay (2010); Roth et al. (2015); Peterson (2014); Cozzi & Ring (2011)
10	Kotlin/Swift Mobile	Kotlin/ Swift	2011/2014	2.x / 6.x	Mobile GIS, field data collection, LBS	Medium	Nivala et al. (2008)
11	Cloud-Native Platforms	Cloud / Distributed	N/A†	N/A†	Scalable spatial processing, cloud storage, APIs	Very High	OGC (2023); vendor documentation for GeoServer, GeoNode and STAC
12	OGC Standards	OGC	1994	OGC API	Interoperability standards	Very High	OGC (2023); Kresse & Danko (2012)

* Maturity ratings are qualitative ordinal assessments derived from four equally weighted criteria: years since first stable release, sustained repository activity in the preceding 24 months, scholarly citation counts, and documented production or enterprise use (applied as of March 2026). † Not applicable — these categories span multiple platforms and standards rather than a single versioned language or ecosystem.

Table 2. The Language-Oriented Geospatial Libraries Ecosystem (LOGLE) framework: representative libraries/platforms, deployment context, and system layer within the software stack (Engine → Data → Processing → Service → Client). Rows 11 and 12 represent cross-cutting categories (Cloud-Native Platforms and OGC Standards).

#	Technology Family	Libraries / Platforms	Deployment	System Layer
1	C/C++ Core Engines	GDAL, PROJ, GEOS, PDAL, GRASS GIS	Engine / Backend	Engine
2	Spatial SQL & Indexing	PostGIS, SpatiaLite, H3	Data / Backend	Data
3	Python Data Science	GeoPandas, Rasterio, PySAL, xarray, OSMnx, Shapely	Application / Backend / Web	Processing

4	R Spatial Statistics	sf, terra, gstat, stars, tmap, spdep	Application / Web	Processing
5	Julia Simulation	GeoStats.jl, ArchGDAL.jl, Rasters.jl	Research / HPC	Processing
6	Java/Scala Enterprise	GeoTools, JTS, GeoMesa, GeoTrellis, Apache Sedona	Backend / Distributed	Service
7	Go Services	tegola, orb, go-geom	Backend / Cloud	Service
8	Rust Systems	geo, proj-rs, gdal-rs	Engine / Systems	Engine
9	JavaScript/TypeScript Web	Leaflet, OpenLayers, CesiumJS, MapLibre GL	Client / Web	Client
10	Kotlin/Swift Mobile	MapLibre Native, Tangram ES	Mobile / Field	Client
11	Cloud-Native Platforms	GeoNode, STAC, Cloud Optimized GeoTIFF, object storage	Cloud / Distributed	Service / Data
12	OGC Standards	WMS, WFS, WCS, OGC API	Cross-layer	Cross-cutting

C and C++ are treated as a unified systems programming ecosystem that provides core geospatial computation libraries such as GDAL, GEOS, and PROJ. Kotlin and Swift are grouped as mobile-oriented ecosystems primarily used in field GIS and mobile mapping applications, often leveraging shared rendering and mapping libraries such as MapLibre Native. JavaScript and TypeScript are treated as a single web ecosystem due to their shared runtime environment and overlapping geospatial visualisation and mapping libraries. Apache Sedona appears in both Java and Scala rows because it is a JVM-based distributed geospatial processing framework supporting both ecosystems. GeoServer is excluded to maintain focus on programmatic libraries rather than standalone server-based GIS platforms. PHP and Ruby are excluded due to limited contemporary use in open-source geospatial computing ecosystems. SQL and Spatial SQL constitute a foundational layer for geospatial data management, enabling persistent storage and declarative querying of spatial data. In contrast, spatial indexing frameworks such as H3 and S2 Geometry operate as a complementary computational layer that supports scalable spatial indexing, efficient data partitioning, and query acceleration within geospatial systems.

4. DISCUSSION

The discussion is organised according to the LOGLE classification and the five-layer architecture summarised in Table 1 and Table 2. The five layers are grouped into four broader themes for discussion purposes: foundational computational engines (Engine layer), spatial data management systems (Data layer), high-level analytical and visualisation ecosystems (Processing and Client layers), and infrastructure and emerging technologies (Service layer and cross-cutting categories). These groupings capture the dependency relationships that exist among geospatial software components.

The LOGLE framework provides a structured taxonomy of open-source geospatial libraries organised by programming language and primary functional role. Library maturity is assessed using the same four criteria set out in Section 2.4. These are years since first stable release, sustained repository activity, scholarly citation counts, and documented production or enterprise use. These criteria inform the ratings in Table 1 but are not presented as quantitative results in this study. The framework also identifies patterns of functional specialisation, language-specific capabilities, and interoperability across the geospatial software ecosystem.

4.1 Python — High-Level Analytical and Machine Learning Workflows

Python occupies a central position in contemporary geospatial research because of its extensive scientific computing ecosystem and the ease with which it integrates with other tools. Libraries such as GeoPandas, Rasterio, and Shapely supply high-level abstractions that rest directly on C and C++ engines. OSMnx supports detailed urban network analysis (Boeing, 2017a, 2017b, 2025). TorchGeo extends the ecosystem into deep-learning applications for remote sensing (Stewart et al., 2021). GeoAI and spatial machine learning workflows are increasingly supported within this ecosystem (Wu, 2026), and domain-specific packages

such as PySNM apply spatial interpolation to soil nutrient mapping in precision agriculture (Velamala & Pant, 2025). In practice, Python is used primarily for analytical workflows, data orchestration, and system integration. Many computationally intensive operations are delegated to underlying C/C++ libraries through language bindings.

4.2 C/C++ — Foundational Computational Engines

C and C++ libraries form the foundational computational layer of the geospatial software stack. GDAL/OGR, GEOS, and PROJ provide the core capabilities for raster and vector processing, coordinate reference system management, and computational geometry (Warmerdam, 2008a). PDAL and Orfeo Toolbox extend this foundation to LiDAR and advanced remote-sensing processing (Grizonnet et al., 2017). These libraries provide high-performance capabilities for format conversion, geometric operations, and data transformation. Higher-level environments such as Python and R access them through binding interfaces and wrapped APIs, providing widely used foundations for computational processing and interoperability across geospatial ecosystems.

4.3 Spatial SQL — Backend Data Management

Spatial databases, principally PostGIS and SpatiaLite, supply the storage, indexing, and querying functions required for persistent geospatial data management (Obe & Hsu, 2015; Ramachandran, 2019). Hierarchical indexing systems such as H3 support efficient spatial indexing and partitioning. This layer functions as the persistent data backbone for desktop, web, and distributed geospatial applications. Analytical and visualisation components routinely read from and write to these systems, making Spatial SQL a critical intermediary between foundational engines and higher-level workflows.

4.4 Java and Scala — Distributed Geospatial Computing

JVM-based frameworks such as Apache Sedona, GeoTrellis, and GeoMesa enable distributed processing of large-scale raster, vector, and spatio-temporal datasets on Apache Spark (Aji et al., 2013; Haynes et al., 2020; Yu et al., 2019). Java provides mature support for server-side geospatial systems through libraries such as GeoTools and the JTS Topology Suite. Scala contributes functional programming constructs that simplify the construction of distributed data pipelines. These systems operate above the data-management layer and interact with both foundational computational engines and storage systems.

4.5 R — Statistical Modelling and Geostatistics

R remains the primary environment for statistical modelling, ecological analysis, and geostatistics within the geospatial domain. Packages including sf, terra, spdep, and gstat support reproducible analytical workflows that combine spatial data handling with advanced statistical methods (Bivand et al., 2013; Gräler et al., 2016; Pebesma, 2018). R is used predominantly for statistical exploration, methodological development, and research that requires tight integration of spatial and statistical techniques. Large-scale production systems and high-throughput pipelines are more commonly implemented in Python or JVM-based environments.

4.6 JavaScript and TypeScript — Interactive Web Visualisation

JavaScript and TypeScript frameworks, including Leaflet, OpenLayers, MapLibre GL, CesiumJS, and deck.gl, provide the dominant tools for web-based geospatial visualisation and interactive mapping (Agafonkin, 2019; Netek et al., 2023; Roth et al., 2015). These libraries are widely deployed in browser-based applications. Client-side computational capacity remains constrained by the browser execution environment. Recent implementations increasingly incorporate WebAssembly modules and GPU-based web APIs to expand the range of operations that can be performed directly in the client.

4.7 Emerging Ecosystems — Rust and Julia

Rust supports both native geospatial projects and bindings to established engines. GeoPolars is a Rust-based project for geospatial data processing, while gdal-rs and proj-rs provide Rust interfaces to the GDAL and PROJ libraries (OSGeo, 2020). These components support memory-safe (Jung et al., 2021) and high-performance geospatial computation within the Rust ecosystem. Julia libraries, including ArchGDAL.jl, GeoStats.jl, and Rasters.jl, target numerical modelling and geospatial simulation (Mallet et al., 2020; Sumner et al., 2023). Rust is optimised for systems-level safety and performance, whereas Julia is optimised for numerical and scientific computation. Both ecosystems are currently at an earlier stage of adoption and community maturity than the more established Python, R, and Java stacks.

4.8 Infrastructure and Mobile-Focused Languages

Go is employed for geospatial backend services, particularly tile servers and spatial APIs, through libraries and frameworks such as Tegola, go-geom, Orb (Golang GIS Community, 2021), and Tile38 (Go Spatial, 2022). Kotlin and Swift, primarily through MapLibre Native (MapLibre Native Project, 2026), support mobile geospatial applications used for field data collection and real-time spatial monitoring. These languages address deployment-oriented and system-level requirements that sit outside the core analytical workflows of Python and R.

4.9 Cross-Language Insights and Interoperability

The LOGLE framework presents geospatial software as a layered and interoperable ecosystem. Foundational engines written in C and C++ and spatial data management systems based on Spatial SQL support higher-level programming environments that include Python, R, Java, and JavaScript. Empirical studies in geospatial computing have concentrated primarily on the Python and R ecosystems (Zhang et al., 2019). Systematic comparative evaluations that include Rust, Julia, and C++ remain relatively limited. Interoperability is achieved through language bindings, shared native libraries, JVM execution environments, and WebAssembly. Performance and scalability characteristics are determined by the chosen execution environment and overall system architecture.

LOGLE is a taxonomy-based classification model. It defines the structural organisation of geospatial ecosystems without assigning quantitative performance metrics. The concentration of scholarly literature on Python and R (210 of 370 included sources) reflects the current maturity and academic visibility of those ecosystems and the relative under-documentation of systems-level and emerging languages. This imbalance does not invalidate the cross-language structure of LOGLE, but it limits the depth of evidence available for Rust, Julia, Go, Kotlin and Swift. Future research can combine the LOGLE classification with standardised benchmarking experiments to examine performance differences across languages and functional roles.

4.10 Illustrative Workflow: Applying the LOGLE Framework in Practice

A multi-language urban flood risk assessment illustrates how the LOGLE framework can be applied in practice. The workflow demonstrates functional specialisation, layered organisation, and interoperability across geospatial software ecosystems (Table 1).

i. Data Ingestion and Backend Storage (Spatial SQL + C/C++)

Vector datasets (building footprints and road networks) and raster datasets (digital elevation models and rainfall surfaces) are stored in a PostGIS database. GDAL/OGR utilities and associated language bindings manage data ingestion. GiST spatial indexing and coordinate transformations performed by PROJ enable efficient spatial querying and data management. This stage corresponds to the foundational computation and storage layers within LOGLE.

ii. Preprocessing and Analytical Processing (Python)

GeoPandas, Rasterio, and Shapely, which rest on GDAL and GEOS, are used to derive terrain attributes from digital elevation models, delineate low-lying areas, and perform spatial joins between infrastructure and flood zones. OSMnx supports the integration of OpenStreetMap networks for accessibility analysis. Python handles data transformation and analytical processing at this stage.

iii. Statistical Modelling and Geostatistics (R)

Spatial datasets are retrieved from PostGIS using sf and DBI/RPostgres interfaces or exported for local analysis. Packages such as sf, terra, gstat, and spdep are applied for spatial autocorrelation analysis, kriging interpolation, and uncertainty estimation. This stage represents statistical and geostatistical modelling within the workflow.

iv. Visualisation and Dissemination (JavaScript/TypeScript)

Interactive maps and dashboards are implemented with Leaflet or MapLibre GL JS. Vector tiles generated by Tegola, a server written in Go (Tegola Project, 2026), support efficient map rendering and web delivery. This stage corresponds to web-based visualisation and dissemination.

v. Performance Optimisation (Emerging Ecosystems)

Julia libraries (Rasters.jl, ArchGDAL.jl) are used for numerical prototyping of computationally intensive workflows. Rust libraries (geo, gdal-rs, proj-rs, t-rex) can be incorporated for memory-safe computation in performance-sensitive tasks (t-rex project, 2021). These emerging ecosystems can be incorporated where specialised computational or implementation requirements arise.

vi. Integrated Interpretation

The workflow illustrates the layered organisation defined by LOGLE. Task allocation follows the characteristic strengths of each ecosystem (C/C++ and Spatial SQL for foundational computation and storage; Python and R for analysis; JavaScript for visualisation). Data exchange relies on standard formats and language bindings. No peer-reviewed benchmark currently exists that measures wall-clock time, memory footprint, and reproducibility of the same multi-stage geospatial pipeline across C/C++, Python, R, Julia and Rust under controlled conditions. The present example therefore serves as a conceptual demonstration; quantitative cross-language evaluation remains an important direction for future work.

5. LIMITATIONS AND FUTURE STUDIES

The proposed LOGLE framework offers a structured classification of open-source geospatial libraries but has inherent limitations. It emphasises widely used programming languages, which may under-represent niche tools and thereby reduce generalisability. Functional categories are derived from documentation that may not fully capture hybrid or rapidly evolving capabilities. Adoption metrics such as GitHub stars and citation counts are imperfect proxies: stars do not necessarily indicate technical uptake, and recently released libraries suffer from citation lag. Maturity ratings reflect the state of the ecosystems at the close of the search period (March 2026), and bibliometric biases can undervalue practitioner-focused ecosystems that have limited academic visibility. Maturity ratings are qualitative and indicative rather than definitive; exact quantitative thresholds were not applied. The proposed framework constitutes a temporal snapshot of a rapidly changing landscape; validation through expert elicitation with GIS practitioners provides only scoped qualitative support. In addition, the literature search may under-represent non-English or niche ecosystems because of its primary reliance on English-language and OSGeo-centric sources. The high proportion of Python/R sources (210 of 370) reflects the current scholarly literature rather than a comprehensive census of operational use. This imbalance is explicitly recognised as a limitation that reduces the depth of evidence available for emerging ecosystems and may have influenced the relative emphasis within the LOGLE categories. The framework's inclusion criteria (minimum of three active libraries with recent repository activity and documented scholarly or technical use) were applied uniformly; the thinner coverage of Rust, Julia, Go, Kotlin and Swift therefore indicates genuine gaps in the published record rather than selective omission.

Future work should empirically validate LOGLE through cross-ecosystem benchmarks of performance, scalability, and memory safety, prioritising metrics such as large-raster throughput, vector processing latency, and memory footprint. Multi-language workflows warrant further study to identify integration bottlenecks and functional overlaps in polyglot pipelines. Periodic reassessment will be required as languages and libraries mature. Emerging approaches, particularly cloud-native geospatial computing and GPU-accelerated spatial analytics, should be systematically documented and evaluated. Finally, non-bibliometric signals (package telemetry, dependency graphs) should be incorporated to strengthen longitudinal relevance and to better link research outputs with operational use.

6. CONCLUSIONS

The LOGLE framework provides a structured, language-oriented classification of the open-source geospatial ecosystem organised by functional roles and technical maturity. It examines twelve programming-language environments (C/C++, Python, R, Julia, Java, Scala, JavaScript, TypeScript, Go, Rust, Kotlin, and Swift), together with SQL/Spatial SQL as a foundational query and data-management layer. These environments encompass both established and emerging tools in geospatial computing. C/C++ engines and SQL extensions constitute the computational and data-management foundation of the ecosystem. Python and R function as the primary environments for geospatial analysis and scientific computing. JVM-based languages support scalable and distributed spatial processing. Rust and Julia provide high-performance, memory-safe capabilities for computationally intensive modelling tasks. Grounded in a systematic literature review, LOGLE offers a coherent reference classification for the analysis of functional specialisation and interoperability within the open-source geospatial stack. The illustrative workflow demonstrates the practical application of this classification. Open-source geospatial libraries offer

cost-effective alternatives to proprietary GIS software and help reduce the financial barrier to sustained environmental and resource monitoring, particularly in developing and underdeveloped countries, thereby contributing to the United Nations Sustainable Development Goals.

AUTHOR CONTRIBUTIONS

Ranga Rao V.: Conceptualization, Methodology, and Supervision.

Pallavi Bhatt: Investigation and Writing – Original Draft.

Prasad N.S.R.: Data Curation and Writing – Review & Editing.

Simhachalam A.: Data Curation and Writing – Review & Editing.

Gahlod N.S.: Validation and Writing – Review & Editing.

Sanjay A. Dhale: Formal Analysis.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

ETHICS STATEMENT

Ethical approval was not required for this study.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

REFERENCES

[1] Agafonkin, V. (2019). *Leaflet: An open-source JavaScript library for mobile-friendly interactive maps*. Retrieved March 15, 2026, from https://leafletjs.com

[2] Aji, A., Wang, F., Vo, H., Lee, R., Liu, Q., Zhang, X., & Saltz, J. (2013). Hadoop-GIS: A high performance spatial data warehousing system over MapReduce. *Proceedings of the VLDB Endowment*, *6*(11), 1009–1020. https://doi.org/10.14778/2536222.2536227

[3] Bivand, R. S., Pebesma, E., & Gómez-Rubio, V. (2013). *Applied spatial data analysis with R* (2nd ed.). Springer. https://doi.org/10.1007/978-1-4614-7618-4

[4] Bivand, R. S., & Wong, D. W. S. (2018). Comparing implementations of global and local indicators of spatial association. *TEST*, *27*(3), 716–748. https://doi.org/10.1007/s11749-018-0599-x

[5] Boeing, G. (2017a). OSMnx: A Python package to work with graph-theoretic OpenStreetMap street networks. *Journal of Open Source Software*, *2*(12), 215. https://doi.org/10.21105/joss.00215

[6] Boeing, G. (2017b). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, *65*, 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[7] Boeing, G. (2025). Modeling and analyzing urban networks and amenities with OSMnx. *Geographical Analysis*, *57*(4), 567–577. https://doi.org/10.1111/gean.70009

[8] Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

[9] Chen, D., Shams, S., Carmona-Moreno, C., & Leone, A. (2010). Assessment of open source GIS software for water resources management in developing countries. *Journal of Hydro-Environment Research*, *4*(3), 253–264. https://doi.org/10.1016/j.jher.2010.04.017

[10] Coetzee, S., Ivanova, I., Mitasova, H., & Brovelli, M. A. (2020). Open geospatial software and data: A review of the current state and a perspective into the future. *ISPRS International Journal of Geo-Information*, *9*(2), 90. https://doi.org/10.3390/ijgi9020090

[11] Cope, M. (2010). A history of qualitative research in geography. In D. DeLyser et al. (Eds.), *The SAGE handbook of qualitative geography* (pp. 3–25). SAGE. https://doi.org/10.4135/9780857021090.n1

[12] Cozzi, P., & Ring, K. (2011). *3D engine design for virtual globes*. A K Peters/CRC Press. https://doi.org/10.1201/9781439865583

[13] Duarte, L., & Teodoro, A. C. (2021). GIS open-source plugins development: A 10-year bibliometric analysis. *Geomatics*, *1*(2), 206–245. https://doi.org/10.3390/geomatics1020013

[14] GeoRust Project. (2026). *GeoRust: An ecosystem of geospatial tools and libraries written in Rust*. Retrieved March 6, 2026, from https://georust.org

[15] Go Spatial. (2022). *Tile38: Real-time geospatial database* (v1.33). Retrieved March 10, 2026, from https://tile38.com/

[16] Golang GIS Community. (2021). *Orb: Geometry operations and spatial indexing for Go* (v0.10). Retrieved March 10, 2026, from https://github.com/paulmach/orb

[17] Goodchild, M. F. (2007). Citizens as sensors: The world of volunteered geography. *GeoJournal*, *69*(4), 211–221. https://doi.org/10.1007/s10708-007-9111-y

[18] Gräler, B., Pebesma, E., & Heuvelink, G. (2016). Spatio-temporal interpolation using gstat. *The R Journal*, *8*(1), 204–218. https://doi.org/10.32614/RJ-2016-014

[19] Grant, M. J., & Booth, A. (2009). A typology of reviews: An analysis of 14 review types and associated methodologies. *Health Information & Libraries Journal*, *26*(2), 91–108. https://doi.org/10.1111/j.1471-1842.2009.00848.x

[20] Grizonnet, M., Michel, J., Poughon, V., Inglada, J., Savinaud, M., & Cresson, R. (2017). Orfeo ToolBox: Open source processing of remote sensing images. *Open Geospatial Data, Software and Standards*, *2*, 15. https://doi.org/10.1186/s40965-017-0031-6

[21] Güting, R. H. (1994). An introduction to spatial database systems. *The VLDB Journal*, *3*(4), 357–399. https://doi.org/10.1007/BF01231602

[22] Haklay, M. (2010). How good is volunteered geographical information? *Environment and Planning B: Planning and Design*, *37*(4), 682–703. https://doi.org/10.1068/b35097

[23] Haynes, D., Huang, Q., & Li, S. (2020). GeoTrellis: A scalable geospatial data processing framework for Apache Spark. *ISPRS International Journal of Geo-Information*, *9*(11), 690. https://doi.org/10.3390/ijgi9110690

[24] Hijmans, R. J. (2023). terra: Spatial data analysis. *Journal of Open Source Software*, *8*(84), 5121. https://doi.org/10.21105/joss.05121

[25] Hijmans, R. J., Bivand, R., Dyba, K., Pebesma, E., & Sumner, M. D. (2020). *terra: Spatial data analysis* (R package). https://doi.org/10.32614/cran.package.terra

[26] Hoffimann, M., Zadrozny, B., & de Souza, R. M. (2018). Geostatistical learning: Challenges and opportunities. *Mathematical Geosciences*, *50*, 451–469. https://doi.org/10.1007/s11004-018-9736-3

[27] Hoyer, S., & Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, *5*(1), 10. https://doi.org/10.5334/jors.148

[28] Huang, Z., Chen, Y., Wan, L., & Peng, X. (2017). GeoSpark SQL: An effective framework enabling spatial queries on Spark. *ISPRS International Journal of Geo-Information*, *6*(9), 285. https://doi.org/10.3390/ijgi6090285

[29] Hughes, J. N., Annex, A., Eichelberger, C. N., Fox, A., Hulbert, A., & Ronquest, M. (2015). GeoMesa: A distributed architecture for spatio-temporal fusion. *Proc. SPIE 9473*, 94730F. https://doi.org/10.1117/12.2177233

[30] Ilich, A. R., Misiuk, B., Lecours, V., & Murawski, S. A. (2023). MultiscaleDTM: An open-source R package for multiscale geomorphometric analysis. *Transactions in GIS*, *27*(4), 1164–1204. https://doi.org/10.1111/tgis.13067

[31] ISO. (2006). *ISO/IEC 13249-3:2006 — SQL multimedia and application packages — Part 3: Spatial*. https://www.iso.org/standard/38651.html

[32] Ježek, J. (2006). Java v open source GIS — GeoTools, GeoServer, uDig. *Geoinformatics FCE CTU*, *1*, 144–148. https://doi.org/10.14311/gi.1.15

[33] Johansson, P., & Harrie, L. (2005). Using JTS for geometric data processing in GIS. *Proceedings of the AGILE Conference 2005* (pp. 221–228).

[34] Jordahl, K., et al. (2020). GeoPandas: Python tools for geographic data. *Journal of Open Source Software*, *5*(52), 2304. https://doi.org/10.21105/joss.02304

[35] Jordahl, K., et al. (2023). *GeoPandas: Python tools for geographic data*. Zenodo. https://doi.org/10.5281/zenodo.8348034

[36] JuliaEarth. (2023). *GeoStats.jl: An extensible framework for geospatial data science and geostatistical modeling*. Retrieved January 10, 2026, from https://juliaearth.github.io/GeoStatsDocs/stable/

[37] Jung, R., Jourdan, J.-H., Krebbers, R., & Dreyer, D. (2021). Safe systems programming in Rust. *Communications of the ACM*, *64*(4), 144–152. https://doi.org/10.1145/3418295

[38] Kanwar, N., Rai, S., & Kuniyal, J. C. (2022). Open-source satellite repository and geographic information system (GIS) for soil resource mapping. In P. K. Shit, P. P. Adhikary, G. S. Bhunia, & D. Sengupta (Eds.), *Soil health and environmental sustainability* (Environmental Science and Engineering). Springer, Cham.

[39] Kitchenham, B., & Charters, S. (2007). *Guidelines for performing systematic literature reviews in software engineering* (Technical Report EBSE-2007-01). Keele University.

[40] Kresse, W., & Danko, D. M. (Eds.). (2012). *Springer handbook of geographic information*. Springer. https://doi.org/10.1007/978-3-540-72680-7

[41] Kumar, N., Singh, S. K., Mishra, V. N., Reddy, G. P. O., & Bajpai, R. K. (2018). Open-source satellite data and GIS for land resource mapping. In G. Reddy & S. Singh (Eds.), *Geospatial technologies in land resources mapping, monitoring and management* (Geotechnologies and the Environment, Vol. 21). Springer, Cham. https://doi.org/10.1007/978-3-319-78711-4_10

[42] Mallet, C., Lantuéjoul, C., & Renard, D. (2020). Geostatistical estimation with GeoStats.jl. *Journal of Open Source Software*, *5*(54), 2481. https://doi.org/10.21105/joss.02481

[43] MapLibre Native Project. (2026). *MapLibre Native: Open-source cross-platform mapping engine*. Retrieved March 2026, from https://maplibre.org/projects/native/

[44] McKinney, W. (2010). Data structures for statistical computing in Python. *Proceedings of SciPy 2010*, 51–56. https://doi.org/10.25080/Majora-92bf1922-00a

[45] Mobasheri, A., Pirotti, F., & Agugiaro, G. (2020). Open-source geospatial tools and their applications in research. *ISPRS International Journal of Geo-Information*, *9*(7), 393. https://doi.org/10.3390/ijgi9070393

[46] Netek, V., Masopust, J., & Komarkova, J. (2023). Leaflet for interactive web maps: Performance and usability. *ISPRS International Journal of Geo-Information*, *12*(3), 112. https://doi.org/10.3390/ijgi12030112

[47] Neteler, M., & Mitasova, H. (2008). *Open source GIS: A GRASS GIS approach* (3rd ed.). Springer. https://doi.org/10.1007/978-0-387-68574-8

[48] Nivala, A. M., Brewster, S., & Sarjakoski, T. (2008). Usability evaluation of mobile maps. *Cartography and Geographic Information Science*, *35*(1), 27–44. https://doi.org/10.1559/152304008783475661

[49] Obe, R. O., & Hsu, L. S. (2015). *PostGIS in action* (2nd ed.). Manning Publications.

[50] Obe, R. O., & Hsu, L. S. (2021). *PostGIS in action* (3rd ed.). Manning. https://doi.org/10.1007/978-1-4842-6618-1

[51] OGC. (2023). *OGC API standards*. Open Geospatial Consortium. https://ogcapi.ogc.org

[52] OSGeo. (2020). *PROJ: Cartographic projections and coordinate transformations* (Version 7.0). Retrieved March 10, 2026, from https://proj.org/

[53] Page, M. J., et al. (2021). The PRISMA 2020 statement. *Systematic Reviews*, *10*, 89. https://doi.org/10.1186/s13643-021-01626-4

[54] Pebesma, E. (2018). Simple features for R. *The R Journal*, *10*(1), 439–446. https://doi.org/10.32614/RJ-2018-009

[55] Pebesma, E., & Bivand, R. (2023). *Spatial data science: With applications in R*. CRC Press. https://doi.org/10.1201/9780429459016

[56] Perkel, J. M. (2019). Why scientists are turning to Julia. *Nature*, *572*, 141–142. https://doi.org/10.1038/d41586-019-02310-3

[57] Peterson, M. P. (2014). Mapping in the cloud. *Cartographic Perspectives*, *78*, 75–89. https://doi.org/10.14714/CP78.1270

[58] Ramachandran, R. (2019). PostGIS for spatial databases and analytics. In *Open source geospatial tools* (pp. 45–62). Springer.

[59] Rey, S. J., & Anselin, L. (2010). PySAL. In *Handbook of applied spatial analysis* (pp. 175–193). Springer. https://doi.org/10.1007/978-3-642-03647-7_11

[60] Rey, S. J., Arribas-Bel, D., & Wolf, L. J. (2020). *Geographic data science with Python*. CRC Press. https://doi.org/10.1201/9780429030595

[61] Rey, S. J., et al. (2022). The PySAL ecosystem. *Geographical Analysis*, *54*(3), 467–487. https://doi.org/10.1111/gean.12276

[62] Rosenbaum, B., et al. (2020). Open-source geospatial tools in R. *Journal of Statistical Software*, *94*(10). https://doi.org/10.18637/jss.v094.i10

[63] Roth, R. E., Ross, K. S., & MacEachren, A. M. (2015). User-centered design for interactive maps. *The Cartographic Journal*, *52*(4), 367–386. https://doi.org/10.1179/1743277414Y.0000000094

[64] Roussel, J.-R., et al. (2020). lidR: An R package for analysis of ALS data. *Remote Sensing of Environment*, *251*, 112061. https://doi.org/10.1016/j.rse.2020.112061

[65] Shekhar, S., Xiong, H., & Zhou, X. (Eds.). (2017). *Encyclopedia of GIS* (2nd ed.). Springer. https://doi.org/10.1007/978-3-319-17885-1

[66] Srivastava, J. (2012). GeoTools: Open source Java GIS toolkit. *Geoinformatica*, *16*(4), 679–700. https://doi.org/10.1007/s10707-012-0145-8

[67] Stewart, A. J., Robinson, C., & Corley, S. (2021). TorchGeo. *ACM SIGSPATIAL*. https://doi.org/10.1145/3474717.3483982

[68] Sumner, M. D., & contributors. (2023). *Rasters.jl: Raster data tools in Julia*. GitHub. https://github.com/rafaqz/Rasters.jl

[69] Tegola Project. (2026). *Tegola: Open source vector tile server*. Retrieved March 2026, from https://tegola.io

[70] Tennekes, M. (2018). tmap: Thematic maps in R. *Journal of Statistical Software*, *84*(6), 1–39. https://doi.org/10.18637/jss.v084.i06

[71] t-rex project. (2021). *t-rex: Vector tile server in Rust*. Retrieved March 10, 2026, from https://github.com/t-rex-tileserver/t-rex

[72] Turton, I. (2008). Geo Tools. In *Open source approaches in spatial data handling* (pp. 153–169). Springer. https://doi.org/10.1007/978-3-540-74831-1_8

[73] Velamala, R. R., & Pant, P. K. (2025). PySNM: An open-source Python package for automated spatial interpolation and soil nutrient mapping for sustainable agriculture — a case study. *Discover Soil*, *2*, 106. https://doi.org/10.1007/s44378-025-00132-6

[74] Warmerdam, F. (2008a). *The GDAL/OGR geospatial data abstraction software library*. https://gdal.org

[75] Warmerdam, F. (2008b). The geospatial data abstraction library. In *Open source approaches in spatial data handling* (pp. 87–104). Springer. https://doi.org/10.1007/978-3-540-74831-1_7

[76] Wu, Q. (2026). GeoAI. *Journal of Open Source Software*, *11*(118), 9605. https://doi.org/10.21105/joss.09605

[77] Yu, J., Zhang, Z., & Sarwat, M. (2019). Spatial data management in Apache Spark. *GeoInformatica*, *23*, 37–78. https://doi.org/10.1007/s10707-018-0330-9

[78] Zaragozí, B., Belda, A., Linares, J., Martínez-Pérez, J. E., Navarro, J. T., & Esparza, J. (2012). A free and open source programming library for landscape metrics calculations. *Environmental Modelling & Software*, *31*, 131–140. https://doi.org/10.1016/j.envsoft.2011.10.009

[79] Zhang, C., et al. (2019). Cloud computing for geospatial big data processing. *ISPRS International Journal of Geo-Information*, *8*(1), 13. https://doi.org/10.3390/ijgi8010013

[80] Zhu, X. X., et al. (2017). Deep learning in remote sensing. *IEEE GRSM*, *5*(4), 8–36. https://doi.org/10.1109/MGRS.2017.2762307